Group-wise Performance Evaluation of Processor Co-allocation in Multi-cluster Systems

John Ngubiri and Mario van Vliet

Nijmegen Institute for Informatics and Information Science
Radboud University Nijmegen
Toernooiveld 1, 6525 ED, Nijmegen,
The Netherlands
{ngubiri, mario}@cs.ru.nl

Abstract. Performance evaluation in multi-cluster processor co-allocation - like in many other parallel job scheduling problems- is mostly done by computing the average metric value for the entire job stream. This does not give a comprehensive understanding of the relative performance of the different jobs grouped by their characteristics. It is however the characteristics that affect how easy/hard jobs are to schedule. We, therefore, do not get to understand scheduler performance at job type level. In this paper, we study the performance of multi-cluster processor co-allocation for different job groups grouped by their size, components and widest component. We study their relative performance, sensitivity to parameters and how their performance is affected by the heuristics used to break them up into components. We show that the widest component us characteristic that most affects job schedulability. We also show that to get better performance, jobs should be broken up in such a way that the width of the widest component is minimized.

1 Introduction

Most research on parallel job scheduling had been focused on single Shared Memory computers, Distributed Memory Multiprocessors and clusters [10][23]. Recently, research work has been extended to computational and data grids [13][27] as well as multi-cluster systems [4][15]. A multi-cluster system is set up by connecting multiple clusters (possibly in different locations) into a bigger computational infrastructure. This is advantageous since it brings together a higher computational power at a lower cost. Jobs submitted to the system can be processed by any of the clusters or a combination of clusters. If multiple clusters are to process a job, the job is broken up into components and co-allocated [7]. The relatively slower inter-cluster communication speed, however, leads to an extended execution time of the co-allocated jobs. This leads to a lower effective utilization of the multi-cluster system. Studies [4] have shown that despite the drawbacks of the slower wide area network, co-allocation is a viable option.

While evaluating performance in parallel job scheduling, the mean value of the measurement metric is mostly used. Previous studies have shown that evaluation using job groups is beneficial. Srinivassan et al. [22] demonstrates how a

deeper understanding of robustness for moldable jobs' schedulers can be obtained using size based groups. Job groups were also used in [24] [25] when comparing performance of Conservative and Aggressive backfilling. More studies by Feitelson [12] employed group-wise analysis for different job streams and measurement metrics to show underlying performance implications.

We use job groups to study the performance of Fit Processor First Served (FPFS) algorithm when scheduling jobs on a multi-cluster system with co-allocation. The jobs are grouped using size, number of components and width of the widest component. We deduce the most influential job characteristic that determines its schedulability. We also compare two heuristics for breaking up large jobs into components as they are being prepared for co-allocation. We also study the sensitivity of the groups' performance on the scheduler and job stream parameters.

We observe that there is a remarkable performance difference among the groups when FPFS scheduler is used. The difference is very small for the FCFS scheduler. We also observe that the width of the widest component is the strongest (of the three) factor that affects the performance (hence schedulability) of a certain job. Finally, we observe that performance can be improved by changing scheduler parameters (within a certain range) and by partitioning the jobs in such a way that the width of the widest component is minimized.

The rest of the paper is organized as follows: We describe our research model in Section 2. In Section 3, we describe the scheduling algorithm and placement policy used. In Section 4, we describe the set up of our experiments. In Section 5, we study the relative performance of the groups and how their performance varies with the parameters of FPFS scheduler. We compare the effect of the component generation heuristics on scheduler performance in Section 6. In Section 7, we study how the proportion of the jobs broken into components affects performance. We discuss related work in Section 8 and make conclusion and suggestions for future work in Section 9.

2 The Research Model

2.1 Multi-cluster set up

We consider a system made up of five homogeneous clusters. The system is served by one queue and one scheduler. The system processes jobs by pure space slicing and is non-preemptive. In case a job has multiple components, the components are co-allocated.

2.2 Job stream

Many supercomputer workloads have been archived [29] and analyzed. Most of them are for single supercomputers not multi-cluster systems which we consider in this study. Logs from the Distributed ASCI Supercomputer [26] were archived in [29] and analyzed in [17]. Co-allocation details however were not included in

the study and the jobs on the different clusters of DAS were archived separately. In this work therefore, we use synthetic workloads. The jobs are online and have exponentially distributed inter-arrival and execution times. These distributions have also been used in previous related work [4][15]. A job's execution time is unknown before it finishes execution. This is because we assume that the user is unable to accurately estimate the duration of his jobs [16].

Job size distribution :
The size of a job is defined by a distribution $D(q)$ where the probability p_i that a job has size i is given by

$$p_i = \begin{cases} \frac{3q^i}{Q} \text{ if } i \text{ is a power of 2} \\ \frac{q^i}{Q} \text{ if } i \text{ is not a power of 2} \end{cases} \quad (1)$$

$D(q)$ $(q < 1)$ is defined over an interval $[n_1,\ n_2]$ $(0 < n_1 < n_2)$. The parameter q is used to vary the mean job size while Q is in such a way that p_i sums up to 1. It favors small jobs and those whose size is a power of 2 which is known to be a realistic choice [11].

Components generation :
Broadly, jobs are divided into two - small and large jobs. Large jobs are broken into components and co-allocated. We define a parameter *thres* to be the size of the largest small job. Any job whose size is bigger than *thres* is broken into components. Multi-component jobs are evenly distributed among the large jobs. If the maximum components a job can have is k, then $2, 3, \cdots k$ component jobs have a proportion $\frac{1}{k-1}$ of the large jobs. To break a job of size s into n components, we make the first $n-1$ components to have width $\lfloor \frac{s}{n} \rfloor$ each while the n^{th} component has width of $s - (n-1)\lfloor \frac{s}{n} \rfloor$. We use two approaches to determine the number of components a certain job will be broken into. We refer to them as *random* and *phased* approach.

a. *Random Approach*
In random approach, we randomly chose the number of components a large job will be broken into. If for example large jobs can have $2, 3, \cdots k$ components, then every large job has a probability of $\frac{1}{k-1}$ of being broken into any of the possible number of components. This is used to represent a situation where the user decides how many components his job should be broken into.

b. *Phased Approach*
In phased approach, the number of components a job has is determined by its size. If large jobs can have $2, 3, \cdots k$ components, we identify job bounds $b_1, b_2, \cdots b_{k-1}$ in such a way that the number of jobs with size range $(thres+1, b_1), (b_1+1, b_2) \cdots (b_{k-2}+1, b_{k-1}), (b_{k-1}+1, s_{max})$ (s_{max} is the largest job size possible) make up $\frac{1}{k-1}$ of the large jobs. Jobs in the size range $(thres+1, b_1)$ are broken into 2 components, those in a size range (b_1+1, b_2) are broken in 3 components and so on. This is used to represent a situation where the system determines the number of components a job should have.

3 The Scheduling Algorithm

We now describe the FPFS scheduling algorithm used in our research.

In FPFS, jobs are queued in their arrival order. When searching for the next job to process, the scheduler starts from the head of the queue and searches deeper into the queue for the first job that fits into the system. In case one is found, it jumps all jobs ahead of it in the queue, gets allocated to the clusters and starts execution. If none is found, the scheduler waits for a job to finish execution or a job to arrive and the search is done again. This may however lead to starvation of some jobs as they are continuously jumped by other jobs from deep inside the queue. This is avoided by limiting the number of times (to $maxJumps$) a job can be jumped while at the head of the queue. After being jumped $maxJumps$ times, no other job is allowed to jump it (and get allocated to clusters) until enough processors has been freed (by terminating jobs) to have it start processing. In this paper, we use FPFS(x) to represent FPFS when $maxJumps = x$.

To map components to clusters, we use the Worst Fit (WFit) placement policy. In this policy, the widest component is placed in the freest cluster (and the smallest component in the busiest cluster). It tends to balance the load among the clusters as well as leaving the free processors as evenly distributed as possible among the clusters.

4 Experimental set up

4.1 Parameters used

We use clusters with 20 nodes. The job stream is generated by $D(0.85)$ on the interval $[1, 38]$. This generates jobs with average size 5.03. Our jobs can be broken into up to 4 components. We consider a case where $thres = 11$ (the effect of the value of $thres$ is studied in Section 7). The jobs have a mean execution time of 10 and mean inter arrival time of 0.64. This leads to a load of 0.786. We chose this load since scheduler performance is more distinct at high loads. We do not model inter-cluster communication.

4.2 Performance evaluation

We use the Average Response Time (ART) as the measurement metric. ART is measured for the entire job stream as well as job groups with in the job stream. Readings are taken at a maximum relative error of 0.05 at 95% confidence interval.

Jobs are grouped by the number of components, size and width of the widest component.

Component wise, jobs are grouped into C_1, C_2, C_3 and C_4 groups. These groups are made up of $1, 2, 3$ and 4 component jobs. At $thres = 11$, C_1 constitutes 89.6% of the jobs and the largest 10% of the jobs are broken into components. The details of the jobs in C_2, C_3 and C_4 depends on the partitioning approach used.

Size wise, jobs are grouped in four groups S_1, S_2, S_3 and S_4 which are size-based quartiles. S_1 consists of jobs with $size = 1$. S_2, S_3 and S_4 groups are made up of jobs in the size range $[2, 3], [4, 7]$ and $[8, 38]$ respectively. S_1, S_2, S_3 and S_4 constitute of $24\%, 26\%, 26\%$ and 24% of the job stream respectively.

Using width of the widest component, we generate groups W_1, W_2, W_3 and W_4 which are widest component based quartiles. The details of the jobs in the widest component based groups are dependent on the partitioning approach. When using the random approach (at $thres = 11$), W_1 and W_2 consist of jobs whose widest component is 1 and 2 while W_3 and W_4 consists of jobs whose width of the widest component is in the ranges $[3, 4]$ and $[5, 19]$. W_1, W_2, W_3 and W_4 jobs constitute $24\%, 26\%, 38\%$ and 12% of the job stream respectively.

5 Job Groups Performance

In this section, we investigate the (relative) performance of the different job groups scheduled by FPFS(0) and FPFS(10). We consider a randomly partitioned job stream ($thres = 11$). Practically, FPFS(0) is the same as FCFS since no job is allowed to jump another. We investigate the variation of performance of job groups with the $maxJumps$ value and deduce the practical implication of the (relative) performance trend.

5.1 Relative performance of job groups

The relative performance of the groups for FPFS(0)/FCFS and FPFS(10) is summarized in Figure 1.

From Figure 1, we observe that (*i*) FPFS(10) performs better than FCFS for all job groups, (*ii*) there is a negligible difference in the different groups' performance for FCFS, (*iii*) there are big performance differences among the job groups when FPFS(10) scheduler is used and (*iv*) big jobs perform poorly.

The improvement in performance as the $maxJumps$ value increase from 0 (FCFS) to 10 can be explained by the global effect of allowing some jobs to jump others and get scheduled. This can be looked at positively and negatively. On the negative side, the jobs that jump may delay the time at which the jobs they jump start execution. This is because the criteria used to allow them jump does not put the execution time into consideration. On the positive side, the jobs that jump actually execute on processors that would be idle. Fishing these jobs from the queue makes it shorter hence lowering the waiting time for the jobs initially behind the jumping job. Since these jobs are used to fill the small remaining gaps, cases of small jobs fragmenting clusters are reduced. The net effect of shortening the queue gives advantage to jobs that have not yet been submitted since they find a shorter queue. Studies by Chiang et al. [6] show that in aggressive backfilling, rarely does a backfilled job delay jobs deep into the queue. This implies that while the disadvantage affects a few jobs at the head of the queue, the advantages go beyond the queue. The advantages outweigh the disadvantages hence a net gain.

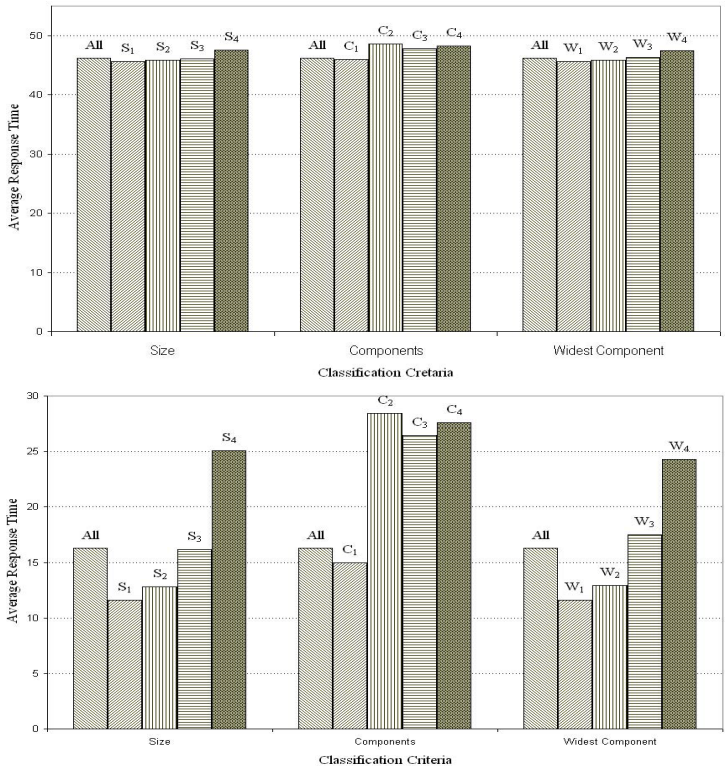

Fig. 1. Performance of different groups for FCFS (top) and FPFS(10) (bottom)

Since FCFS does not allow jobs to jump others, a job at the head of the queue blocks all others behind it until there are enough free processors in the system. There is therefore minimal performance difference. The slightly better performance of small jobs can be attributed to the fact that they are less likely to wait for long while at the head of the queue. Since FPFS allows mostly small jobs to jump and get scheduled before others ahead of it, there is a good performance for small jobs and a poor performance for large ones. We observe a direct relationship between performance and size and performance and width of the widest component. Jobs in C_2, C_3 and C_4 basically belong to S_4. The poor performance of C_2 shows the effect of size and widest component affects the performance trend in component based trends.

The system cannot alter the size of the job since we assume they are rigid. However, depending on the partition decisions made, the width of the widest component can be altered. This implies that a user who is interested in improving performance should concentrate on the partition approach and should aim at having jobs with narrow components.

The effect of the width of the widest component is caused by the way free processors are distributed in the clusters as dictated by the placement policy. Since WFit places the widest component in the freest cluster, it distributes the free processors among the clusters as evenly as possible. It is therefore hard to get a big block of free processors in a single cluster to process a wide component. Since FPFS allows jobs that can fit to jump those which cannot fit, the terminating jobs do not necessarily create the required processors as the jumping jobs reduce them further. This causes relative starvation for jobs with wide components.

5.2 Performance variations with $maxJumps$

We now investigate the variation of performance with $maxJumps$ for the different job groups. We use the size and width of the widest component partitions to illustrate our results (components follow the same trend) and summarize the results in Figure 2.

From Figure 2, we observe that the different job groups have a similar performance trend. Increasing $maxJumps$ leads to an improvement in performance. The rate of improvement is high at low $maxJumps$ values. At high $maxJumps$ values ($maxJumps \geq 10$), there is a minimal improvement in performance. Large jobs perform poorer compared to small jobs.

Increasing $maxJumps$ gives more jumping opportunities for small jobs. At very high $maxJumps$ values, small jobs are processed immediately they arrive (mean execution time is 10, therefore, the minimum mean response time is 10).

5.3 Load-wise Implication

We now investigate the implication of the deviation in performance for the different groups. We do this by comparing amount of the load in each group. We summarize the comparative numerical and load composition in Table 1.

Table 1. Numerical and load contribution of the different groups (partition approach - random, $thres = 11$)

Criteria					
Size			Widest component		
Group	Number(%)	Load(%)	Group	Number(%)	Load(%)
S_1	24.88	5.08	W_1	24.88	5.08
S_2	25.64	11.49	W_2	25.64	11.49
S_3	25.20	24.13	W_3	22.98	23.45
S_4	24.28	59.30	W_4	26.50	59.98

From Table 1, we observe that there is a skewed relationship between the numerical and load wise contribution of the different job groups. This is in

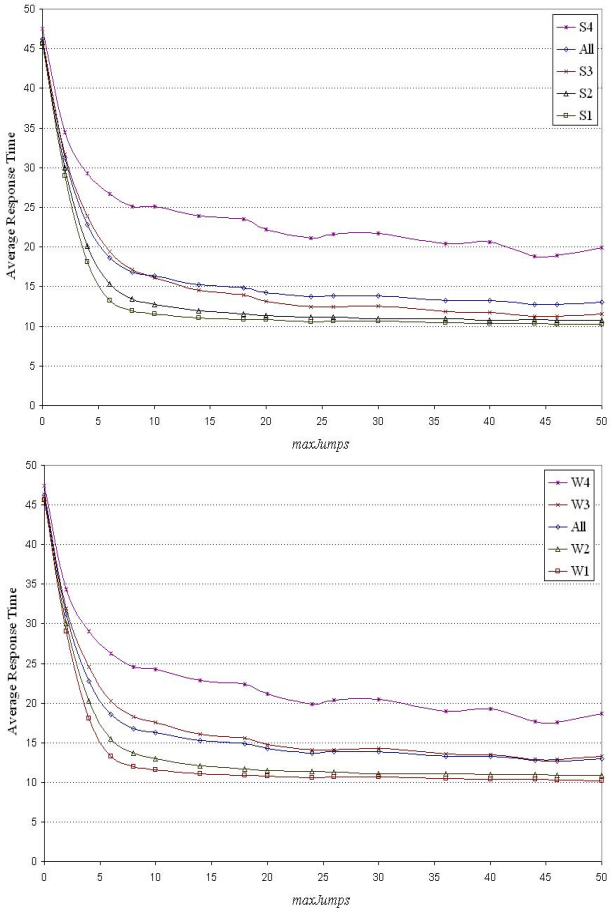

Fig. 2. Performance variations with $maxJumps$ for job groups grouped by size (top) and widest component (bottom)

agreement with previous studies on workload characteristics [19]. Groups S_4 and W_4 register worst performance and constitute more than half of the load in the system. This implies that a big portion of the workload actually perform worse than the average. Since a job's contribution to the average response time is independent of its load, the numerical minority of poor performing jobs is not adequately implied by the ART metric.

6 The Effect of the Partitioning Format

In Section 5, we only used the random partitioning approach. We observed that the width of the widest component highly influence performance and can be altered by the partitioning scheme. In this section, we compare the random and

phased partitioning approaches at $thres = 11$. We summarize the performance trend in Figure 3. We use only size based groups when making performance evaluation. This is because the partitioning approach does not influence the distribution of jobs in S_1, S_2, S_3 and S_4. Component based groups, for phased partitioning, are analogous to size based groups with different job proportion. We only show S_1 and S_4 which are the extremes. S_2 and follow the same trend.

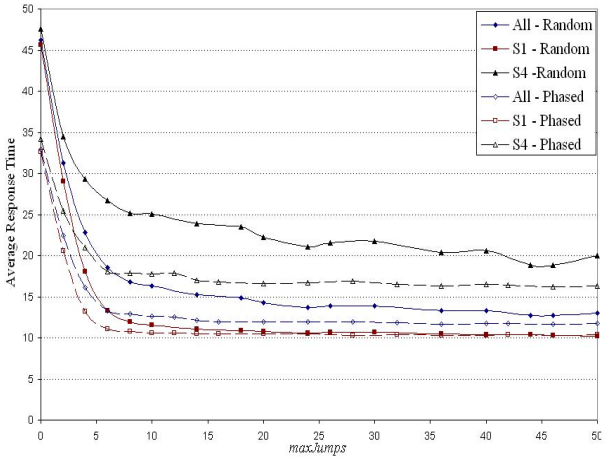

Fig. 3. Performance variations of selected groups with $maxJumps$ for job streams partitioned using the random and phased approach($thres = 11$)

From Figure 3, we observe that there is an improvement in performance for jobs in S_1 and S_4. There is also a reduction in the difference between S_4 and S_1 jobs. The improvement is also registered in for FCFS.

The improvement in performance can be explained by the change in widest component width distribution. Since phased partition approach breaks bigger jobs into more components, the widest components gets lower hence easier to pack.

This implies that performance can be improved by improving the scheduling approach as well as improving the partitioning approach. Since the difference between S_1 and S_4 is smaller in a phased partitioned job stream, it shows us that the improvement in performance comes with the improvement in fairness.

7 The Effect of *thres*

From Section 6, we observe that it is actually necessary to break up large jobs and co-allocate them. We took a case where $thres - 11$. We investigate the effect of $thres$ value on scheduler performance. We show the trend in Figure 4.

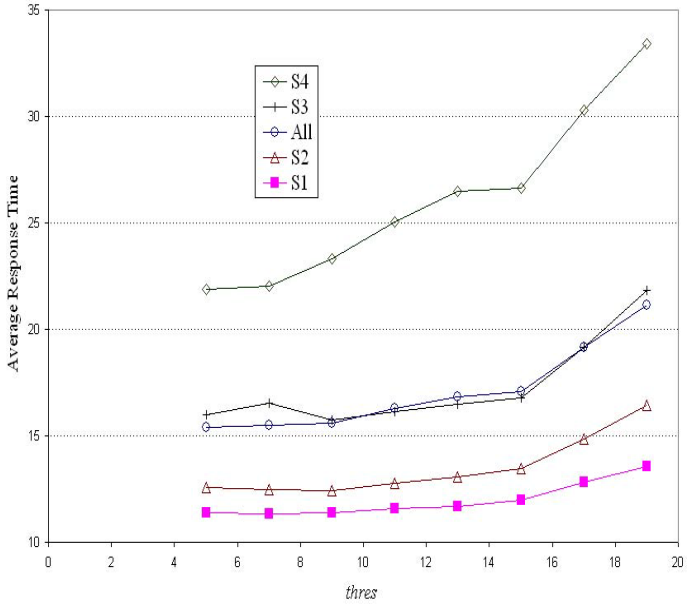

Fig. 4. Performance variations of FPFS(10) with *thres* for a randomly partitioned job stream

From Figure 4, we observe that increasing *thres* leads to poor performance. the deterioration of performance is higher for large jobs than for small jobs.

Small jobs can easily be fished from the queue by the scheduler. This implies that whether *thres* is low or high, they have high chances of being scheduled. Having a high *thres* however leads to poor component placement and adversely impacts on the entire job stream. Lowering *thres* is therefore beneficial to the entire job stream but mostly to the large jobs. At high *thres* values, the scheduler is more unfair. The value of *thres* therefore needs to be kept low in th interest of performance and fairness.

8 Related Work

Multi-cluster systems have attracted a sizeable amount of research of recent. Some multi-cluster systems have also been set up for research in performance analysis, co-allocation and the general field of parallel processing. These include the Distributed ASCI Supercomputer (DAS) [26] located in The Netherlands. Earlier work on DAS was reported by Bal et al [1]. Specific work on co-allocation on DAS was done by Jones [15], Bucur [4] and Bucur and Epema [5], [2], [3].

Many schedulers have been proposed for parallel job scheduling [10]. For space slicing cases like the one we are considering in this research, backfilling has been the most popular. Backfilling seeks to allow small jobs to be processed

before they reach the head of the queue. This is under a condition that they do not pose a disadvantage to the jobs they jump. This is therefore catered for in the conditions the job has to satisfy to be allowed to jump. Conservative backfilling [18] allows the job to jump others if it will not delay the reservation time of any job in the queue. Aggressive backfilling [20] on the other hand allows a job to jump if it will not affect the reservation time of the job at the head of the queue. Backfilling basically allows jobs to jump and get processed without giving a setback to the jumped jobs. Work by Shmueli and Feitelson [21] employs backfilling but by ensuring that the job picked is the one which can offer the best utilization of the processor hole available. It however brings in an extra consideration of determining how far deep in the queue the scheduler should go while searching for the job that offers best results. This is only possible if enough information about the jobs (execution time in this case) is known; in some cases, they are not known [8]. Lack of knowledge of some parameters restricts what the scheduler can do. Backfilling cannot be employed in cases where the runtime of the jobs is unknown. This because the scheduler lacks the basis on which reservations can be made. Lack of runtime knowledge can be caused by the inability of the user to accurately estimate the job runtime [16]. In such a case, a job can be allowed to jump so long as it can fit in the available processor hole. This is done in the FPFS scheduler. Starvation is controlled by limiting the number of times a job at the head of the queue is jumped.

The choice of the measurement metric for parallel job scheduling evaluation has to be taken with a lot of care [14]. Some metrics may have different implications depending on the circumstance. While average waiting time and average response time can have a similar performance for dedicated processing, they don't in a time sliced / preemptive cases.

Grouping jobs by their characteristics and evaluate how they perform helps in getting a deeper understanding of the scheduler performance [12]. Srinivasan et al. [22] use job groups to get a deeper understanding of scheduler robustness while scheduling moldable jobs. Deeper comparative studies between Conservative and Aggressive backfilling were studied in [25][24] by studying performance of job groups in the job stream.

In multi cluster systems, studies have been done to study and improve scheduler performance. Jones [15] focused on scheduling techniques and how they are affected by network characteristics like latencies. Bucur and Epema investigated the performance of co-allocation in different scenarios. This involved queue prioritization [5], job structures [2] and scheduling policies [3].

9 Conclusions and Future Work

We have investigated the group-wise performance of processor co-allocation in multi-cluster systems. We use the FPFS scheduler. We have observed that increasing $maxJumps$ improves performance for the jobs that jump and those which are jumped. There is however a higher level of unfairness as $maxJumps$ is increased. We have also observed that the performance can be improved by

using a superior scheduler, a batter partitioning strategy as well as breaking up a bigger proportion of jobs. When partitioning, the aim should be put at having a low width of the widest component.

Our work also opens up more avenues for research. Phased partitioning has shown an improvement in performance. There is a need to investigate a possibility of a better partitioning heuristic. This is due to the fact that the widest component is actually reducible. There is also a need to investigate the how the job characteristics like average size and moldability affect scheduler performance.Breaking up jobs into components creates a packing benefit but also comes with a communication overhead. Communication leads to an increase in the execution time of the job. Different jobs have different intensity of communication and different multi-cluster systems have different local area and wide area communication speeds. There is therefore need for detailed study of communication based effects and the extent of their effects on scheduler performance. Finally, there is a need to investigate performance behavior using smaller partitions (beyond the quartiles used here) in order to get a deeper understanding of the inter-job performance behavior.

References

1. Bal et al. The Distributed ASCI Supercomputer Project. Operating Systems Review 34(4), pp 76 - 96, 2000.
2. A. I .D Bucur and D. H. J Epema. The Influence of the Structure and Sizes of Jobs on the Performance of Co-allocation, In proceedings of JSSPP'00, LNCS 1911, pp 154 - 173, 2000.
3. A.I.D Bucur and D.H.J Epema. The Performance of Processor Co-allocation in Multicluster Systems. In proceedings of the 3^{rd} IEEE/ ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), pp 302-309, 2003.
4. A.I.D. Bucur, Performance Analysis of Processor Co-allocation in Multicluster Systems. PhD Thesis, Delft University of Technology, Delft, The Netherlands, 2004.
5. A.I.D Bucur and D.H.J Epema, Local versus Global Schedulers with Processor Co-allocation in Multicluster Systems. In proceedings of JSSPP'02, LNCS 2537, pp 184-204, 2002.
6. S. H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. In proceedings of JSSPP'02, LNCS 2537, pp. 103 – 127, 2002.
7. K. Czajkowski, I.T. Foster and C. Kasselman, Resource Co-allocation in Computational Grids, In proceedings of the 8^{th} IEEE International Symposium on High Performance and Distributed Computing, California, USA, pp 37 – 47, 1999.
8. J. Edmonds. Scheduling in the Dark. In proceedings of the 31^{st} Annual ACM Symposium on Theory of Computing, pp 179 - 188, 1999.
9. D.G. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik and P. Wong, Theory and Practice in Parallel Job Scheduling. In proceedings of JSSPP'97, LNCS 1291, pp 1-34, 1997.
10. D.G. Feitelson, L. Rudolph and U. Schwiegelshohn, Parallel Job Scheduling A Status Report. In proceedings of JSSPP'04, LNCS 3277, pp 1 - 16, 2004.
11. D.G. Feitelson and L. Rudolph. Towards Convergence of Job Schedulers for Parallel Supercomputers. In proceedings of JSSPP'96. LNCS 1162, pp 1 - 26, 1996.

12. D. G. Feitelson, Metric and Workload Effects on Computer Systems Evaluation. Computers 36(9), pp. 18 - 25, 2003.
13. I. Foster and C. Kasselman, The Grid: Blue Print for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, USA, 1999.
14. E. Frachtenberg and D.G.Feitelson. Pitfalls in Parallel Job Scheduling Evaluation. In proceedings of the JSSPP'05, LNCS 3834, pp 257 - 282, 2005.
15. W.M. Jones, Improving Parallel Job Scheduling Performance in Multi-clusters Through Selective Job Co-allocation, PhD dissertation, Clemson University, Clemson, South Carolina, USA, 2005.
16. C. B. Lee, Y.Schwartzman, J. Hardy and A. Snavely, Are User Runtime Estimates Inherently Inaccurate? In proceedings of JSSPP'04, LNCS 3277, pp 253-263, 2004.
17. H. Li, D. Groep, and L. Walters, Workload Characteristics of a Multi-cluster Supercomputer. In proceedings of JSSPP'04, LNCS 3277, pp. 176 - 193, 2004.
18. L. Lifka, The ANL/IBM SP Scheduling System, In proceedings of JSSPP'95, LNCS, 949, pp 295-303, 1995
19. J. Moreira P. Pattnaik H. Franke, J. Jann. An Evaluation of Parallel Job Scheduling for ASCI Blue-Pacific. In proceedings of the IEEE/ACM Supercomputing Conference SC'99. Portland, Oregon, USA., 1999.
20. A. W. Mualem and D.G. Feitelson, Utilization, Predictability, Workloads and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, IEEE Transactions in Parallel and Distributed Systems, 12(6) pp 529-543, 2001.
21. E. Shmueli and D. G. Feitelson, Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling. In proceedings of JSSPP'03, LNCS 2862, pp. 228 – 251, 2003.
22. S. Srinivasan, S. Krishnamoorthy and P. Sadayappan. A Robust Scheduling Technology for Moldable Scheduling of Parallel Jobs. In Proceedings of the IEEE International Conference on Cluster Computing, pp 92 – 99, 2003.
23. E. Strohmaier, J. J. Dongarra, H. W. Meuer and D. Simon. Recent Trends in the Marketplace of High Performance Computing. Journal of Parallel Computing, Vol 31, pp 261 - 273, 2005
24. S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan. Selective Reservation Strategies for Backfill Job Scheduling. In proceedings of JSSPP'02, LNCS 2537, pp 55-71, 2002.
25. S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan. Characterization of backfilling Strategies for Parallel Job Scheduling. In proceedings of the 2002 International Conference on Parallel Processing Workshops, pp 514 - 520, 2002.
26. The Distributed ASCI Supercomputer. *www.cs.vu.nl/das2*.
27. The Global Grid Forum. www.gridforum.com
28. The Mesquite Software inc : The CSIM 18 Simulation Engine Users Guide.
29. The Parallel Workloads Archive *http://www.cs.huji.ac.il/labs/parallel/workload/logs.html*.